NAME

Font::TTF::Manual – Information regarding the whole module set

INTRODUCTION

This document looks at the whole issue of how the various modules in the TrueType Font work together. As such it is partly information on this font system and partly information on TrueType fonts in general.

Due to the inter-relation between so many tables in a TrueType font, different tables will make expectations as to which other tables exist. At the very least a font should consist of a `head` table and a `maxp` table. The system has been designed around the expectation that the necessary tables for font rendering in the Windows environment exist. But inter table dependencies have been kept to what are considered necessary.

This module set is not meant as a simple to use, mindless, font editing suite, but as a low-level, get your hands dirty, know what you are doing, set of classes for those who understand the intricacies (and there are many) of TrueType fonts. To this end, if you get something wrong in the data structures, etc. then this module set won't tell you and will happily create fonts which don't work.

At the time of writing, not every TrueType table in existence has been implemented! Only the core basic tables of TrueType 1.0 (i.e. no embedded bitmap tables, no postscript type tables, no OpenType tables and no GX tables) have been implemented. If you want to help by implementing another table or two, then please go ahead and send me your code. For a full list of tables, see Font::TTF::Font.

Design Principles

PERL is not C++. C++ encourages methods to be written for changing and reading each instance variable in a class. If we did this in this PERL program the results would be rather large and slow. Instead, since most access will be read access, we expose as much of the inner storage of an object to user access directly via hash lookup. The advantage this gives are great. For example, by following an instance variable chain, looking up the `yMax` parameter for a particular glyph becomes:

```
$f->{'loca'}{'glyphs'}[$glyph]{'yMax'}
```

Or, if we are feeling very lazy and don't mind waiting:

```
$f->{'loca'}{'glyphs'}[$f->{'cmap'}->ms_lookup(0x41)]{'yMax'}
```

The disadvantage of this method is that it behoves module users to behave themselves. Thus it does not hold your hand and ensure that if you make a change to a table, that the table is marked as *dirty*, or that other tables are updated accordingly.

It is up to the application developer to understand the implications of the changes they make to a font, and to take the necessary action to ensure that the data they get out is what they want. Thus, you could go and change the `yMax` value on a glyph and output a new font with this change, but it is up to you to ensure that the font's bounding box details in the `head` table are correct, and even that your changing `yMax` is well motivated.

To help with using the system, each module (or table) will not only describe the methods it supports, which are relatively few, but also the instance variables it supports, which are many. Most of the variables directly reflect table attributes as specified in the OpenType specification, available from Microsoft (<http://www.microsoft.com/typography>), Adobe and Apple. A list of the names used is also given in each module, but not necessarily with any further description. After all, this code is not a TrueType manual as well!

Conventions

There are various conventions used in this system.

Firstly we consider the documentation conventions regarding instance variables. Each instance variable is marked indicating whether it is a **(P)**rivate variable which users of the module are not expected to read and certainly not write to or a **(R)**ead only variable which users may well want to read but not write to.

METHODS

This section examines various methods and how the various modules work with these methods.

read and read_dat

Before the data structures for a table can be accessed, they need to be filled in from somewhere. The usual way to do this is to read an existing TrueType font. This may be achieved by:

```
$f = Font::TTF::Font->open($filename) || die "Unable to read $filename";
```

This will open an existing font and read its directory header. Notice that at this point, none of the tables in the font have been read. (Actually, the `head` and `maxp` tables are read at this point too since they contain the commonly required parameters of):

```
$f->{'head'}{'unitsPerEm'}
$f->{'maxp'}{'numGlyphs'}
```

In order to be able to access information from a table, it is first necessary to `read` it. Consider trying to find the advance width of a space character (U+0020). The following code should do it:

```
$f = Font::TTF::Font->open($ARGV[0]);
$snum = $f->{'cmap'}->ms_lookup(0x0020);
$sadv = $f->{'hmtx'}{'advance'}[$snum];
print $sadv;
```

This would result in the value zero being printed, which is far from correct. But why? The first line would correctly read the font directory. The second line would, incidently, correctly locate the space character in the Windows cmap (assuming a non symbol encoded font). The third line would not succeed in its task since the `hmtx` table has not been filled in from the font file. To achieve what we want we would first need to cause it to be read:

```
$f->{'hmtx'}->read;
$sadv = $f->{'hmtx'}{'advance'}[$snum];
```

Or for those who are too lazy to write multiple lines, `read` returns the object it reads. Thus we could write:

```
$sadv = $f->{'hmtx'}->read->{'advance'}[$snum];
```

Why, if we always have to read tables before accessing information from them, did we not have to do this for the `cmap` table? The answer lies in the method call. It senses that the table hasn't been read and reads it for us. This will generally happen with all method calls, it is only when we do direct data access that we have to take the responsibility to read the table first.

Reading a table does not necessarily result in all the data being placed into internal data structures. In the case of a simple table `read` is sufficient. In fact, the normal case is that `read_dat` reads the data from the file into an instance variable called `' dat'` (including the space) and not into the data structures.

This is true except for the `glyph` class which represents a single glyph. Here the process is reversed. Reading a `glyph` reads the data for the glyph into the `' dat'` instance variable and sets various header attributes for the glyph (`xMin`, `numContours`, etc.). The data is converted out of the variable into data structures via the `read_dat` method.

The aim, therefore, is that `read` should do the natural thing (read into data structures for those tables and elements for which it is helpful — all except `glyph` at present) and `read_dat` should do the unnatural thing: read just the binary data for normal tables and convert binary data to data structures for `glyphs`.

In summary, therefore, use `read` unless you want to hack around with the internals of glyphs in which case see Font::TTF::Glyph for more details.

update

The aim of this method is to allow the various data elements in a `read` font to update themselves. All tables know how to update themselves. All tables also contain information which cannot be *updated* but is new knowledge in the font. As a result, certain tables do nothing when they are updated. We can, therefore, build an update hierarchy of tables, with the independent tables at the bottom and `Font` at the top:

```
      +--loca
      |
glyf--+--maxp
      |
      +---+--head
          |
hmtx------+--hhea

cmap-----OS/2

name--

post--
```

There is an important universal dependency which it is up to the user to keep up to date. This is C<maxp/numOfGlyphs> which is used to iterate over all the glyphs. Note that the glyphs themselves are not held in the C<glyph> table but in the C<loca> table, so adding glyphs, etc. automatically involves keeping the C<loca> table up to date.

Creating fonts

Suppose we were creating a font from scratch. How much information do we need to supply and how much will `update` do for us?

The following information is required:

```
$f->{'loca'}{'glyphs'}
$f->{'head'}{'upem'}
$f->{'maxp'}{'numGlyphs'}   (doesn't come from $f->{'loca'}{'glyphs'})
$f->{'hmtx'}{'advance'}
$f->{'post'}['format'}
$f->{'post'}{'VAL'}
$f->{'cmap'}
$f->{'name'}
```

Pretty much everything else is calculated for you. Details of what is needed for a glyph may be found in Font::TTF::Glyph. Once we have all the information we need (and there is lots more that you could add) then we simply

```
$f->dirty;          # mark all tables dirty
$f->update;         # update the font
```

AUTHOR

Martin Hosken <http://scripts.sil.org/FontUtils>. (see CONTRIBUTORS for other authors).

LICENSING